CMPSC 465 Spring 2024 | Data Structures & Algorithms
Mehrdad Mahdavi and David Koslicki | Worksheet 2

Wednesday, January 24, 2023

1. Growth Rate. Sort the following functions based by their growth rate. (You may assume all logarithms have base 2.)

(a) $(\sqrt{2})^{\log n}$
(b) n^2
(c) $n!$
(d) $(\log n)!$
(e) $(\frac{3}{2})^n$
(f) $(\log n)^2$
(g) n^3
(h) $log(n!)$
(i) 2^{2^n}
(j) $n^{\frac{1}{\log n}}$
(k) $\log\log n$
(l) $n2^n$
(m) $n^{(\log\log n)^2}$
(n) $2^{2^{2n+1}}$
(o) $2^{\log n}$
(p) $2^{\sqrt{2\log n}}$
(q) $\sqrt{\log n}$

Solution: Note that here the symbol $<$ represents a comparison of growth rates, not actual function values.

$$n^{\frac{1}{\log n}} < \log\log n < \sqrt{\log n} < (\log n)^2 < 2^{\sqrt{2\log n}} < (\sqrt{2})^{\log n} < 2^{\log n} <$$
$$\log(n!) < n^2 < n^3 < (\log n)! < n^{(\log\log n)^2} < (\tfrac{3}{2})^n < n2^n < n! < 2^{2^n} < 2^{2^{2n+1}}$$

- $n^{\frac{1}{\log n}} < \log\log n$. Taking the log of both sides gives $\frac{1}{\log n}\cdot\log n < \log\log\log n$, or $1 < \log\log\log n$.
- $\log\log n < \sqrt{\log n}$. Substituting $k = \log n$: $\log k < k^{\frac{1}{2}}$. Any polynomial dominates any logarithm.
- $\sqrt{\log n} < (\log n)^2$. Substituting $k = \log n$: $k^{\frac{1}{2}} < k^2$.
- $(\log n)^2 < 2^{\sqrt{2\log n}}$. Taking the log of both sides gives $2\log\log n < (\sqrt{2\log n})(\log_2 2)$. Substituting $k = \log n$ gives $2\log k < (2k)^{\frac{1}{2}}$. Any polynomial dominates any logarithm.
- $2^{\sqrt{2\log n}} < (\sqrt{2})^{\log n}$. Taking the log of both sides gives $(\sqrt{2\log n})(\log_2 2) < (\log n)(\log_2 2^{\frac{1}{2}})$. Substituting $k = \log n$ gives $(2k)^{\frac{1}{2}} < \frac{1}{2}k$. Higher degree polynomial dominates.
- $(\sqrt{2})^{\log n} < 2^{\log n}$. With equal exponents, the higher base dominates.
- $2^{\log n} < \log(n!)$. $2^{\log_2 n} = n$ while $\log(n!) = \Theta(n\log n)$.
- $\log(n!) < n^2$. $\log(n!) = \Theta(n\log n)$ while $n^2 = \Theta(n^2)$.
- $n^2 < n^3$. Higher degree polynomial dominates.
- $n^3 < (\log n)!$. Taking the log of both sides gives $3\log n < \log((\log n)!)$. Substituting $k = \log n$ gives $3k < \log(k!) = \Theta(k\log k)$.
- $(\log n)! < n^{(\log\log n)^2}$. Taking the log of both sides gives $\log((\log n)!) < (\log\log n)^2(\log n)$. Substituting $k = \log n$ gives $\log(k!) < k(\log k)^2$. $\log(k!) = \Theta(k\log k) < k(\log k)^2$.

- $n^{(\log\log n)^2} \;<\; (\frac{3}{2})^n$. Taking the log of both sides gives $(\log\log n)^2(\log n) < n\log(\frac{3}{2})$. Any polynomial dominates any logarithm.
- $(\frac{3}{2})^n \;<\; n2^n$. $\frac{3}{2} < 2$, higher base exponent dominates.
- $n2^n \;<\; n!$. Taking the log of both sides gives $\log n + n\log(2) < \log(n!) = \Theta(n\log n)$.
- $n! \;<\; 2^{2^n}$. Taking the log of both sides gives $\log(n!) < 2^n\log 2$. $\log(n!) = \Theta(n\log n)$ is dominated by any exponential.
- $2^{2^n} \;<\; 2^{2^{2n+1}}$. $2^{2^{2n+1}} = 2^{2^{2n}\cdot 2} = (2^{2^{2n}})^2$. Substituting $k = 2^{2^{2n}}$ gives $k < k^2$.

2. Find run time. How long does the recursive multiplication algorithm (shown below) take to multiply a non-negative n-bit number by a non-negative m-bit number? Justify your answer.

Algorithm 1 multiply(x,y)

```
Input: An n-bit integer x and an m-bit integer y, where x,y ≥ 0
Output: Their product
if y = 0 then
    return 0
end if
z = multiply(x, ⌊y/2⌋)
if y = even then
    return 2z
else
    return x + 2z
end if
```

Solution: Assume we want to multiply the n-bit number x with the m-bit number y. The algorithm terminates after at most m recursive calls; at each call y is halved and the number of digits is decreased by one. Each recursive call requires dividing y by 2 (rounded down), which can be done in $O(1)$ time by shifting. Checking the parity of y can be done in $O(1)$ by checking its rightmost bit (if the rightmost bit is 0, y is even). Similarly, $2z$ can be calculated by shifting. Note that z is the result of multiplying x by $\lfloor \frac{y}{2} \rfloor$ and thus has $O(\log(x \cdot \lfloor \frac{y}{2} \rfloor)) = O(n+m)$ digits. Therefore, the sum $x+2z$ takes $O(n+m)$ time. Counting all of these operations, each recursive call takes $O(n+m)$ time, and the total running time is $O(m\cdot(n+m))$.

3. Computing Factorials. Consider the problem of computing $N! = 1\times 2\times\cdots\times N$.

1. If N is a b-bit number, how many bits long is $N!$ (Θ notation suffices)?
2. Consider the simple algorithm to compute $N!$ that does the multiplication in sequence, $1\times 2\times 3\times \ldots \times N$. Analyze its running time.

Solution:

1. We know that the number of bits of $N!$ is $\Theta(\log_2(N!))$ which we know from the previous worksheet that it is $\Theta(N\log N) = \Theta(b\cdot N)$ since $b = \Theta(\log_2 N)$.
2. We can compute $N!$ naively as follows:

```
factorial (N)
    f = 1
```

```
for i = 2 to N
    f = f · i
return f
```

Running time: we have N iterations, each one multiplying an $(N \cdot b)$-bit number (at most) by a b-bit number. Using the naive multiplication algorithm, each multiplication takes time $O(N \cdot b^2)$. Hence, the running time is $O(N^2 b^2)$.

4. Fibonacci growth. The Fibonacci numbers $F_0, F_1, F_2 \ldots$ are defined by the recurrence $F_0 = 0, F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$. In this problem we will confirm that this sequence grows exponentially fast and obtain some bounds on its growth.

(a) Use induction to prove that $F_n \geq 2^{0.5n}$ for $n \geq 6$.

(b) Find a constant $c > 0$ such that $F_n \geq 2^{cn}$ for all $n \geq 3$. Show that your answer is correct.

(c) Find the maximum constant $c > 0$ for which $F_n = \Omega(2^{cn})$?

Solution:

(a) Base cases: $F_6 = 8 \geq 8 = 2^{6/2}$ and $F_7 = 13 \geq 11.314 \approx 2^{7/2}$.

Inductive Step: For $n \geq 7$, we have

$$F_{n+1} = F_n + F_{n-1} \geq 2^{n/2} + 2^{(n-1)/2} = 2^{(n-1)/2}(2^{1/2} + 1) \geq 2^{(n-1)/2}2 \geq 2^{(n+1)/2}$$

as desired.

(b) $F_3 = 2$ and $2^{3c} = 8^c$. $2 \leq 8^c$ is true for any value of $c \leq \frac{1}{3}$. Therefore, $F_n > 2^{cn}$ for all $n \geq 3$ is true for any constant $c \leq \frac{1}{3}$. Notice how reducing n_0 from 6 to 3 also loosened the lower bound from $2^{\frac{1}{2}n}$ to $2^{\frac{1}{3}n}$. This implies that to prove a very tight lower bound, we will need to pick a sufficiently large n_0.

(c) The argument in part (a) holds as long as we have, in the inductive step, $2^{c(n-1)}(2^c + 1) \geq 2^{c(n+1)}$.
Substituting $k = 2^c$ gives $k^{(n-1)}(k+1) \geq k^{(n+1)}$.
Then we divide both sides by $k^{(n-1)}$ to get $k + 1 \geq k^2$, or $k^2 - k - 1 \leq 0$.
The maximum solution to this quadratic is $k = 2^c \leq \frac{1+\sqrt{5}}{2}$, the golden ratio.